The Envoy Framework: An Open Architecture for Agents

MURUGAPPAN PALANIAPPAN, NICOLE YANKELOVICH, GEORGE FITZMAURICE, ANNE LOOMIS, BERNARD HAAN, JAMES COOMBS, and NORMAN MEYROWITZ
Institute for Research in Information and Scholarship (IRIS)
Brown University

The Envoy Framework addresses a need for computer-based assistants or agents that operate in conjunction with users' existing applications, helping them perform tedious, repetitive, or time-consuming tasks more easily and efficiently. Envoys carry out missions for users by invoking envoy-aware applications called operatives and inform users of mission results via envoy-aware applications called informers. The distributed, open architecture developed for Envoys is derived from an analysis of the best characteristics of existing agent systems. This architecture has been designed as a model for how agent technology can be seamlessly integrated into the electronic desktop. It defines a set of application programmer's interfaces so that developers may convert their software to envoy-aware applications. A subset of the architecture described in this paper has been implemented in an Envoy Framework prototype.

Categories and Subject Descriptors: A.1 [**Introduction and Survey**]; C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*distributed applications*; D.2.2 [**Software Engineering**]: Tools and Techniques—*software libraries*; H.1.2 [**Models and Principles**]: User/Machine Systems—*human factors*; H.3.4 [**Information Storage and Retrieval**]: Systems and Software—*current awareness systems (selective dissemination of information—SDI)*; H.4.1 [**Information Systems Applications**]: Office Automation; K.6.3 [**Management of Computing and Information Systems**]: Software Management—*software development*

General Terms: Design, Human Factors, Management

Additional Key Words and Phrases: Application programmer interface, user agent

1. INTRODUCTION

Electronic computer-based agents may significantly improve users' ability to manage information; however, their impact to date has been minimal. This

Authors' addresses M. Palaniappan, Aldus Corporation, Seattle, WA 98104, muru@aldus.celestial.com; N. Yankelovich, Sun Microsystems Laboratories, Billerica, MA 01821, nicole.yankelovich@east.sun.com; G. Fitzmaurice, University of Toronto, M5T 1N4 Ontario, Canada, gf@dgp.toronto.edu; A. Loomis, Go Corp., Foster City, CA, 94404, anne_loomis@go.com; B. Haan, Siemens Nixdorf, Cambridge, MA 02142, bhaan@sni-usa.com; J. Coombs, IRIS, Brown University, Providence, RI 02912, jhc@iris.brown.edu; N. Meyrowitz, Go Corporation, Foster City, CA 94404, nkm@go.com

paper examines a spectrum of approaches previously applied to the implementation of agents. Based on our analysis of the strengths of these various approaches, we have designed and prototyped a new open architecture for agents. We refer to our implementation of agents as *Envoys*. The Envoy architecture, which provides the focus for this paper, represents a practical approach to the wide-spread integration of agent technology with standard end-user applications.

Research at IRIS has been focused on hypermedia technology and information retrieval [5, 34]. We have learned that these tools alone are not sufficient to help users who routinely work in an information-rich environment. They only solve a segment of the problems users encounter. Users still must devote substantial time to actively seeking and sorting information—browsing, issuing queries, or following links to find pertinent data. They still have a difficult time distinguishing new information from old information. And they still have trouble figuring out who else is working in their computing environment and what information other users are adding, deleting, or changing.

In the future, envoy technology can potentially have a dramatic effect on the way people interact with electronic information. Once the technology matures, both the jobs of sifting through incoming information and actively seeking specific information can be delegated to Envoys. By shifting to a *delegation model*, people would be able to dispatch Envoys to monitor data sources likely to contain useful information, freeing them from the time-consuming and repetitive aspects of these sifting and searching tasks. If users, instead of Envoys, were to engage in such proactive behavior, their time would be entirely consumed by searching for information. In the end, they may also be less effective at finding critical information. As part of their experimental evaluation of the Clipping Service project, Gifford and Fracomano [16] have demonstrated that an active, user-programmed agent is more effective in retrieving information of interest to the user than the user is in actively seeking out the same information.

In the technology that we have prototyped, an Envoy takes on the burden of waiting, watching, and searching, while the user only need to specify missions and to review results—which are preclassified according to the person's specifications—as they arrive. People can focus, therefore, more of their time and energy on primary tasks such as writing reports, analyzing data, or creating on-line publications, confident that when relevant information is available, *it* will find them.

1.1 Overview of Envoy Framework and Prototype

We have adopted a user interface metaphor to help users understand and interact effectively with Envoys. A user specifies a *mission* for the Envoy by interacting with an "envoy-aware" application. We call envoy-aware applications *operatives* because they are responsible for actually carrying out missions on behalf of the user. Once the user specifies a mission, the Envoy plays the role of coordinator by scheduling, tracking, and dispatching all missions the user has specified (Figure 1). As the user's representative, the Envoy

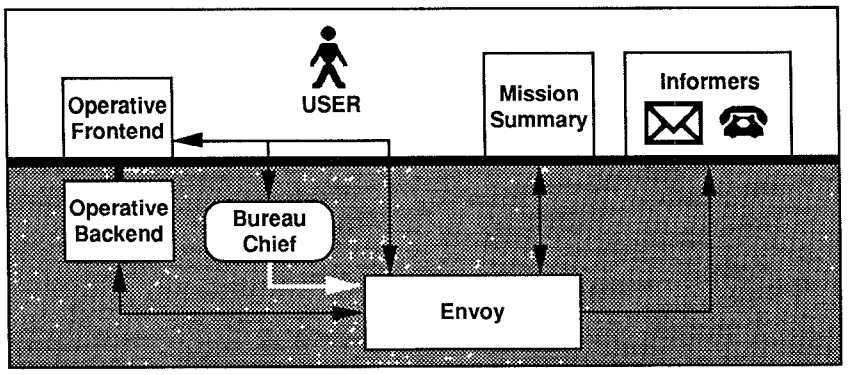


Fig. 1. Overview of Envoy Framework components.

handles all communication with the operatives. If the user has specified an information-gathering mission, then the operative assigned to the mission lets the Envoy know when new information is available to report to the user. In turn, the Envoy notifies the user, selecting a communication channel from a set of envoy-aware applications called *informers*. Once notified of mission results, either with a brief *message* or a *short report*, the user can opt to see an *interactive report*. The Envoy stores interactive reports generated by operatives. To view an interactive report, the Envoy passes the data to the operative responsible for carrying out the mission, giving users the ability to manipulate the mission results using the native application interface. At any time, the user may display a *Mission Summary*, which provides a comprehensive list of all the user's active missions and all the reports generated by the operatives responsible for those missions.

When an application developer first introduces a new operative or informer into the environment, it is registered with a *Bureau Chief*. For every local-area network (LAN) there is one Bureau Chief, which maintains a record of all envoy-aware applications in the environment as well as a record of each user's personal Envoy.

1.2 Operatives and Informers

There are three types of operatives: *monitor operatives* that monitor data continuously over a span of time, *scheduled operatives* that execute tasks at scheduled times, and *combined operatives* that can do both of these two functions.

In our prototype, we have upgraded a file system browser to adhere to our Envoy protocol, and we are nearly finished upgrading a full-text retrieval application. The first, a UNIX™ File System Browser/Monitor (referred to below simply as "the Browser"), is a monitor operative. With this application, users can browse the UNIX file hierarchy and track changes to any file systems in their LAN. The second, Document Search, is a combined opera-

tive. This information-retrieval application for UNIX file systems allows users to search the full text and attributes of any regular UNIX file. Users may opt to either monitor the file system for any new or updated information or specify queries to be run at scheduled times.

We believe that a wide range of desktop applications will be suitable for developers to upgrade to operatives; however, there are also some that do not lend themselves to working on missions for the user. Text and graphics editors, for example, are best used interactively; we do not expect these types of applications to become envoy aware. On the other hand, any type of information retrieval application acting on collections such as file systems, full-text databases, bibliographic databases, and news feeds would be ideally suited for information-gathering missions on behalf of the user. In addition, any applications that manipulate large or complex data could be controlled by an Envoy. For example, if time-consuming calculations or database operations are involved, a user could specify the task to be done and delegate the mission to the Envoy for execution during off-peak hours.

The second type of envoy-aware applications, Informers, can be any signals already in the operating environment, such as beeps, flashing menu bars, or alert boxes. They can also be any end-user communication program such as the UNIX "write" facility, electronic mail, the InternetExpress service [28], or a FAX application. Our prototype allows users to decide how they would like to be informed of mission results by giving them the choice between two informers; electronic mail or alert boxes.

Before describing our Envoy architecture in detail, we discuss the benefits of adopting a delegation model of information retrieval, and we examine previous work related to agents, culling out the significant characteristics embodied by each different approach.

2. BENEFITS OF SHIFTING TO A DELEGATION MODEL

The shift from sifting and sorting behaviors to specifying and delegating behaviors can bring about major gains for people who routinely work with information. In this section, we describe how Envoys, as we envision them in their maturest form, would reduce labor and benefit users by operating within the framework of a delegation model. For example, we envision that Envoys will automate the processes of sifting through incoming information, monitoring existing data sources, carrying out repetitive tasks, and deferring the execution of delegated tasks. They will also provide users with a number of services including individualized reporting and flexible notification of mission results.

Throughout this section, we use the term Envoy as a short-hand for referring to the Envoy/operative team.

2.1 Automation

With a mature Envoy system, people could stay informed of just those topics or events that matter to them. They would not have to attend as carefully to

notices of general distribution. For example, if a user is interested in any information posted on network news about the Hypertext '93 conference, the user currently has to sift through all the conference and hypertext news postings to find relevant articles, which may or may not exist. In many cases, the percentage of relevant information contained in a data source is so low that the cost of an individual monitoring this information for a small number of relevant items is high. Because some of these large data sources are time consuming to sift through, many people do not take the time to keep abreast of topics which interest them. If users could conveniently delegate the task of sifting through incoming data to an Envoy, they would be freed from having to examine these data sources manually.

Once Envoys are well established, people should soon become aware that notices of general distribution do not need to be generated so frequently. If, for example, an engineer finds it useful to track all new specification documents created by his division, such tracking can be performed without requiring that a human generate regular reports. In fact, tracking could be initiated and terminated without any explicit coordination with other members of the organization. This capability changes the base assumption from "we should broadcast *all* information that people may find useful" to "let individuals delegate their Envoys to acquire routine information."

If Envoys are able to monitor a wide range of electronic data sources, they may be of great benefit to users working in a networked environment who wish to keep abreast of *changing* information. One of the most frequent problems reported to us by users of our *Intermedia* hypermedia system is the difficulty in knowing what information has changed and by whom. With today's file systems, which we consider an important source of full text and other data, it is hard to know if anyone else is currently working with the same data or if anyone else has changed information since it was last examined.

As Envoys become more and more able to monitor shared information sources, they will increasingly be relied on as a communication aid between collaborators. Team members will be able to keep abreast of additions, deletions, and changes other team members are making in the shared electronic work space. The onus can be placed on the Envoy to track group activity, rather than require each team member to carefully report on current activities. In this way, new or updated information will automatically find the user.

Sometimes a person may prefer not to monitor a data source, but rather to search it at discrete intervals. For example, it may be too expensive to monitor a commercial on-line database service. Searching the database periodically is more economical, but requires repetition. Doing the same task over and over can be tedious, time consuming, and not the most effective use of a person's time. For example, logging on to a full-text database such as MedLine once a week to check for new items of interest can be a waste of time if no new relevant articles are available. On the other hand, the cost of missing relevant information is high, making the task necessary, if not always fruitful. People often wish to delegate these types of repetitive tasks to

assistants if possible. By providing an easy mechanism to specify such a task and how often that task should be carried out, Envoys will eventually allow people to delegate a variety of repetitive tasks in the domain of the electronic work environment.

Envoys also have the potential to benefit people who would like to carry out a task now that time constraints dictate must be done in the future. For example, a manager might know that the newest sales figures will be entered into the company database a week from now. While the thought occurs, the manager could formulate a query requesting the latest figures, delegating responsibility to the Envoy for running the query on the appropriate day. By specifying the task in the present, the person would be able to delegate responsibility to the Envoy for executing the task in the future. This deferred execution capability could also be helpful when a task involves using a scarce resource or doing something that would be disruptive to others at the current time. For example, it might be possible to re-sort a large database now, but everyone else in the office accessing the database would be disrupted. Better to do the time-consuming, disruptive operation during off-peak hours by specifying the task now and delegating it to an Envoy to carry out later.

2.2 Services

In addition to the automation of sifting, monitoring, repetitive, and deferred tasks, Envoys will eventually be able to provide users with a set of services. The most significant of these involves reporting mission results. Consider a person trying to discover who has changed what in a jointly authored electronic collection, or consider a person repeating the same search for new information of interest. These people often find it difficult to remember the previous state of the environment. For example, a person searching an article database might spot an interesting title, begin to read the article, and then recognize the article as one that was previously read. Or a person might list the files in a shared directory by date, hoping to find all the new additions, but not quite remembering the exact data the directory was last examined.

In conjunction with envoy-aware end-user applications, Envoys provide individualized reporting services. When monitoring a shared workspace or an electronic data source, Envoys provide users with a clear indication of how the state of the electronic environment has changed since their last report. People will no longer have to rely on their memory of what they have and have not seen in the past, nor will they have to waste time unintentionally reviewing old information.

In shifting to a delegation model, it is essential to keep users abreast of their delegated missions and to alert them as soon as new information is available. Each user's Envoy serves as a center for communications and has the ability to relay information to the user through a variety of channels (envoy-aware communication applications). Consequently, the bearers of important information (envoy-aware end-user applications) can contact the communications hub, which can then absorb the burden of trying alternative routes to reach the user who needs the information. In this way, Envoys

direct information to people who regularly travel from home to office to laboratory and beyond.

3. BACKGROUND

Our approach to integrating Envoys with standard end-user applications builds on the work of others. Apple Computer has produced a video* exploring a futuristic system called the Knowledge Navigator [3, 22]. In this video, a character responds to voice-activated commands and performs tasks for the user, such as searching for data in external databases and screening incoming telephone calls. As a preliminary step towards achieving a full-fledged Knowledge Navigator, Apple has demonstrated a system called Guides that includes video personalities that present the user with different perspectives on a topic and suggest alternate routes through educational material [7, 27].

The Knowledge Navigator and the Guides system are reminiscent of the guide concept envisioned at Atari as part of their Electronic Encyclopedia project [33]. An encyclopedia guide would act as a user's personal agent by finding, filtering, and explaining information. Users could select as their guide an agent with a particular personality. This personality, for example, a Renaissance scholar, Albert Einstein, or a Disney character, would dictate the type, complexity, scope, and slant of the information presented to the user.

The Corporation for National Research Initiatives (CNRI) has introduced another vision. They have proposed a scheme for creating a National Digital Library System, which would include tireless Knowbots (knowledge robots) responsible for sorting, analyzing, maintaining, and finding information in a network-wide electronic library [19].

3.1 Artificial Intelligence Approaches

Most of the visionary conceptions of agents and a large number of the implemented systems involve varying degrees of intelligence on the part of the agent.

3.1.1 *Rule-Based Systems.* The most common mechanism for endowing agents with intelligence is to base their behavior on rules. LIZA [10, 15], PAGES [17], ISM [29], Object Lens [21], and Information Lens [23] are examples of systems that incorporate a rule-based approach. If the conditions of a rule are satisfied, then its actions are executed by the agent.

PAGES, for example, targets forms migration. Each user has an agent and a database of forms. Each form includes fields, data, and the specification of one or more rules. Using agents, the system provides a mechanism for users to route forms to other users on the network. The user interface to the forms,

**Project 2000—A Knowledge Navigator* is available from Apple Computer @1-800-627-0230.

the agent processes, and the forms database all can be distributed across the network. By uniquely identifying agents, PAGES allows users to route and share forms across both a local- and a wide-area network. Users create new forms by subclassing instances of existing forms and modifying the layout, fields, and rules associated with the forms. Users must learn an interpreted rule language to modify an existing rule or write a new rule.

In contrast to PAGES, Object Lens helps users construct rules by providing them with structured templates for specifying rules. These templates alleviate the need for users to learn the syntax of a programming language to specify the behavior of agents. The technique of using templates to define agents and create rules for them provides users with a consistent way of specifying the behavior of agents. Whether the user would like an agent to sort incoming mail, retrieve information from a database, or track the progress of a project, the mechanism for specifying these tasks is standardized.

The primary advantage of a rule-based approach is the flexibility rules provided in extending the functionality of agents. On the other hand, users who are focused on primary tasks such as writing reports, reading mail, or looking for information may be unable or unwilling to construct the sort of rules that make these systems useful.

3.1.2 *Systems Based on "Society of Mind" Theory.* A number of agent systems are based on Minsky's "society of mind" theory [25], which postulates a divide-and-conquer theory of human intelligence. The mind, according to the theory, consists of a swarm of communicating agents, each running in parallel, attending to a single aspect of a problem. These agents are organized hierarchically and can be grouped together for the purpose of completing a task more complicated than any one agent can tackle individually.

This theory has inspired several researchers to implement systems based on multiple cooperating agents. NeuralAgents [1] and the agents within the Playground system [12] provide two such examples. Both of these systems use cooperating agents for developing simulations where actions routinely occur in parallel.

While appealing in the simplicity of individual agents and their rules, in practice, the Society-of-Mind approach leads to some formidable complexities. For example, will users be able to figure out which agents to combine together to carry out a complex task, or will users be able to determine which agents can work in parallel?

3.1.3 *Adaptive Systems.* A third potential AI approach to agents, which may place less burden on users than rule-based approaches, involves agents that display intelligence by learning from experience and adapting their behavior to their user's habits, needs, likes, and dislikes. There are many examples of adaptive systems in the learning, diagnostic, and reasoning domains. Ellman [11] provides many examples of intelligent systems that learn by example.

One system that specifically includes an adaptive agent is the Office

CLERK [24]. In this system, the CLERK learns the bulk of how to accomplish a task by example. That is, the user demonstrates some sequence of actions to the CLERK. To then explain conditions or iterations to the CLERK, the user can either provide written instructions or perform more instances of the task for the CLERK to learn the desired behavior.

Another system that has some adaptive characteristics is called COKES [20]. In this rule-based system, each agent, though identical in nature to others, has a knowledge base appropriate to the needs, authority, and responsibility of its particular user. Specialized servers have knowledge of general interest and maintain centralized control over changes to this knowledge. All agents and servers are equipped with inference engines for communicating and cooperating with each other. For example, an agent would be able to support a newly appointed manager by first identifying members of a project staff and later, operationally, by prompting the manager for authorization to distribute timely requests to various project members for submission of their contributions to the project.

Basing agent behavior on learning strategies is a promising approach since it alleviates the need for users to think about and create rules. In practice, however, the time and expertise necessary to create knowledge bases which provide domain knowledge that agents can use as a basis for learning is still impractical.

3.2 Specialization Approach

Much less ambitious than any of the AI approaches, the specialization approach involves imbedding a special-purpose agent within an application to allow users to automate a particular task. For example, to find the electronic address of a user on the Internet, a person must know about and issue queries to three or four different lookup services that have been independently developed. Of course, each of these services has a different user interface. The Directory Access Service [9] helps users automate the lookup process. In response to a query, an agent forwards the query to the various lookup services scattered around the Internet. The agent then collects and formats the responses, sending the results of the multiple queries to the user in an electronic mail message.

There are many on-line database-retrieval services [6, 8, 14] that provide some degree of query automation. For example, the Alert service allows users to specify an Associated Press (AP) news wire query. This query can then be registered with an agent of sorts. The agent will run the query either on a daily or a weekly basis, depending on the time interval specified by the user. Based on the user's preference, the agent will notify the user about the results of the query through postal mail or electronic mail.

This kind of specialized agent imbedded within an application is helpful to users; however, the approach is not at all general. Currently, users can only enlist assistants to find information about other users with the Directory Access Service, and users can only expect agents to trigger AP news wire queries and return results with the Alert Service. Although these services

allow users to monitor information, users have to learn different interfaces, notification styles, and reporting techniques for each service.

3.3 Watchdog-Based Approach (Monitoring Tools)

Considerably more general than the special-purpose agents described above, a number of systems incorporate agents to help monitor computer environments and notify users when certain events occur. UNIX Crontab [32], Sun's Calendar Manager [30], and Sun's SunNet Manager [31], described below, are representative examples of such systems. The Rand Intelligent Terminal Agent (RITA) [2] and the Network Event Manager [4] are other examples of systems that fall into the category of monitoring tools. Most of these tools are targeted primarily towards system administrators and programmers.

The UNIX system scheduling tool, called Crontab, can be considered an agent of sorts. It allows users to define when and how often they want commands to be carried out. The system, rather than the user, monitors the system clock and executes commands at the times users have specified in a *crontab file*. This file provides a consistent mechanism for users to register commands for the system to carry out repetitively. Crontab's delayed execution feature enables users to be less time conscious about performing tasks.

Sun's Calendar Manager application provides a portion of Crontab's functionality with a significantly more intuitive user interface. Like Crontab, the Calendar Manager monitors the system clock watching for times specified by the user. Unlike Crontab, the Calendar Manager only performs a single type of delayed task—sending a calendar entry to the user on a specified date.

More elaborate than both Crontab and the Calendar Manager, the SunNet Manager is targeted towards monitoring and managing a network. The network manager comes with a set of small applications or agents designed to collect network statistics. A single interface to all these agents is provided for users to specify how and when reports should be generated as well as how and when to report results. As agents run and collect statistics, the data are stored in a central database. When a reporting condition is met, the user may be notified by a signal (blinking icon or bell) or by electronic mail. Once notified, the user can request to see the data in a number of forms, including bar charts and strip charts, or can specify that the statistics be passed as input to another program. System administrators can use the network manager to monitor a site's network, or they can write new managers that coordinate a completely different set of agents.

While the functions currently performed by SunNet Manager agents are narrow in scope, the architecture is extensible. The system allows application programmers to add new managers and agents into the SunNet Manager framework. Adding these tools to the framework enhances their value since the framework provides a single, consistent mechanism for triggering the agents, reporting results, and notifying users of results.

The SunNet Manager is appealing in many respects; however, it suffers from a number of problems. A large obstacle to creating new sets of agents is that only tools that can share the same user interface can be integrated with a single manager. Even the interface to fairly simple tools would have to be

redefined to fit into the manager framework. Highly interactive tools such as database- or full-text-retrieval applications would probably be extremely difficult, if not impossible, to convert into agents.

In general, the watchdog or monitoring tools approach to agents suffers from a lack of generality. Monitoring the environment can be quite useful, but it only addresses one class of tasks that agents can potentially perform for users.

3.4 Desktop-Based Approach

Unlike the SunNet Manager approach, in the desktop-based approach, agent functionality is integrated with a set of full-fledged end-user applications. The main difference involves the framework in which agents operate. In the SunNet Manager, agents are integrated with the SunNet Manager core. In the desktop-based approach, agents are integrated into a computer's main operating environment or desktop. New Wave Agent [18] exemplifies this approach. In the New Wave environment, application developers implement a defined set of protocols to make their applications agent aware. This scheme allows users of agent-aware applications to take advantage of agent functionality with a moderate amount of effort on the part of the application developer.

New Wave Agent is a tool for automating tasks that users frequently perform. For example, an agent could start a database access application, download specified information into a spreadsheet, generate a graph of the data in the spreadsheet, copy the graph into a text document, and mail the text document to a group of users. This type of agent behavior is specified by example. Users turn on a recording feature and perform the desired set of actions interactively. This produces a script that can be edited by the user. Figure 2 shows a simple script that opens, empties, and then closes the user's waste basket. Represented on the desktop by the script document, the task can be scheduled using a calendar. Users interact with the calendar to specify how often the task should be carried out and at what time intervals.

The seamless integration of the agent functionality into the desktop environment provides users with the power to automate repetitive tasks that they normally would have to carry out manually. Since tasks can be defined by example, the cognitive overhead of learning a scripting language is reduced. Users need to gain only enough familiarity with the language to be able to make modifications to the scripts. In addition, the calendar provides a natural and intuitive mechanism for users to schedule agent tasks.

The New Wave Agent approach capitalizes on users' existing skills and knowledge of applications, requiring them to learn a minimal amount to take advantage of agents. In theory, the "difficult" parts of tasks will be recorded for users automatically in the scripts. There are, however, a number of problems with the approach. First, New Wave Agents are both single user and single tasking. If an agent task is scheduled to run while a user is working, the agent will take over the processor to carry out the task, only returning control to the user when the task is completed or when the user explicitly interrupts the agent. Basing agent tasks on scripts is also

```
FOCUS OFFICE
Open_Waste# = 1
ON ERROR DO Cant_Select WasteBasket
SET ERROR ON
SELECT WASTE_BASKET
SET ERROR OFF
  IF Open_Waste# = 1
    OPEN
  ENDIF
FOCUS WASTE_BASKET
EMPTY
  IF Open_Waste# = 1
    CLOSE
  ENDIF
END
ENDTASK

PROCEDURE Cant_Select_WasteBasket
  IF SYS_ERROR( ) = 197
    Open_Waste# = 0
    RETURN
  ENDIF
  MESSAGE Cancel# "Can't open
    the Waste Basket"
    + STR(SYS_ERROR( ))
    & EXCLAMATION_POINT OK
  END
ENDPROG
```

Fig. 2. Example of a New Wave Agent script that opens, empties, and then closes the waste basket. The script is written in a special-purpose task language.

problematic in that scripts may become obsolete as the user's environment is modified. For example, the script that downloads data into a spread sheet will fail if the user has inadvertently moved the spreadsheet application to a different directory.

3.5 Characteristics and Capabilities of Agent Systems

Each of the systems described above incorporates features and characteristics associated with the agent concept. In analyzing these systems, we have extracted what we believe are the most significant characteristics embodied in this work.

Functionally, the systems described above provide examples of agents that are *event* or *time triggered*. A task might be triggered by the user, by a certain prerequisite event occurring, or by the time of day. Some of the systems provide for *delayed execution* of tasks or commands, allowing the users to specify what they would like done and when in the future they would like it to be done. The most sophisticated systems are *resilient to failure*, display *adaptive behavior*, and provide for *pipelining of tasks*. A resilient system enhances users' confidence by not losing track of missions entrusted to an agent when machines are down or when network failures occur. An

adaptive system changes its method of carrying out a task based on the user's response to previous interaction with the agent, and in the systems that support pipelining, one agent's task can be triggered by the completion of another agent's task.

Although not demonstrated in any of the systems described above, we believe two other characteristics will substantially contribute to the usability of an agent system. First, users must be provided with some mechanism for *tracking* an agent's activity, just as they would want frequent status reports on the activity of a human assistant. In addition, agents should provide users with *flexible notification* options, giving users a variety of potential communication channels from which to choose.

For people to continue to use an agent system, the architecture must be *extensible*. Either users themselves have to be able to add new behaviors, or there must be a uniform mechanism for application developers to enhance the functionality of agents. In some cases, extensibility may be achieved by *interagent communication.* By having agents work in conjunction with one another, tasks beyond the ability of any single agent can be accomplished. Extensibility may be made easier by creating a *decentralized* agent architecture. Decentralization, or partitioning agent functionality into component parts, will help to make an agent system manageable. Extensions or upgrades can be accomplished by modifying or replacing modules rather than one large monolithic program.

We believe a prime characteristic contributing to the usability of an agent system is *seamlessness.* If users are going to accept agents, they must fit seamlessly into the computing environment, working in conjunction with the tools a person commonly uses. This type of integration also necessitates a reasonable degree of *consistency*. If agents can perform a variety of tasks in a user's standard computing environment, it is important that users have a uniform way to specify these tasks. This mechanism must also involve low cognitive overhead. A *simple interface* for specifying tasks will allow users to concentrate on their work rather than concentrate on interacting with the agent. Two characteristics—*distributed architecture* and *heterogeneity*—are essential if agents are to exist in an networked environment. In such an environment, users, data, and applications are spread across multiple machines and file servers, not always of the same type. Users, who normally have access to a variety of resources on the LAN, will expect agents to perform missions throughout this entire domain. As WANs offer more resources with the same ease of access as LANs users will expect agents to carry out missions in this broader domain. In designing the Envoy framework we attempt to embody many of these characteristics, as shown in Figure 3.

4. ENVOY USER INTERFACE

In designing the user interface for the Envoy prototype, we adopted a fundamental principle: Envoys must save time for users, not make extra work for them. Envoys only require users to learn a few simple concepts. To help users carry out essential tasks, Envoys fit seamlessly into the user's

Agent Systems Comparison Chart	Adaptive	Seamless	Consistent	Extensible	Resilient	Decentralized	Distributed	Heterogeneous	Agent tracking	Notify options	Inter-agent comm.
PAGES			y	y(6)		y	y	y(1)			
Object Lens			y	y(6)		y				y	
ISM (Mailtrays)			y	y(6)		y					y
NeuralAgent			y			y					y
Playground			y			y					y
COKES	y(5)		y	y		y	y	y(1)			y
Direct Acc Ser					y	y	y	y			
Dialog Alert			y	y(4)	y		y	y		y	
UNIX Crontab		y	y	y(2)				y(1)			
Sun Calender Mgr			y		y		y			y	
Sun Net Mgr				y		y		y(1)			
RITA		y			y		y				
Network Evt Mgr				y	y	y	y	y(1)			
New Wave Agent		y	y	y					y(3)		
Envoy Framework		y	y	y	y	y	y	y(1)	y	y	

Fig. 3. Summary of characteristics and capabilities of select agent systems.

(1) Works across only UNIX-based systems.
(2) Partial. Crontab cannot be easily extended to do anything other than launch executables.
(3) New Wave Agent calendar is limited to scheduling information. No mission status is provided.
(4) Extensible to the extent that Dialog can add databases to their search services.
(5) Adaptive only in the sense that the agent behavior changes as knowledge in the various knowledge bases is updated.
(6) Extendible to the extent that new classes can be defined, but it is unclear how agents access external applications.

computing environment, taking advantage of the applications a user already employs.

4.1 Using Envoys

To interact with an Envoy, a user first specifies a task or mission using the standard user interface of an operative. Specifying a mission can be as simple or as complex as ordinary interaction with the application. The rules defining a mission are implicitly specified by the user's interaction with an operative. With Envoys, users never explicitly write or edit rules using a programming or scripting language. For example, a system administrator might want to monitor a confidential directory to detect any unauthorized access to the data. To do so, the administrator would interact with the Browser, mark the desired directory, and register the request to monitor the directory with the Envoy (Figure 4).

In another example, a user might open Document Search, enter the phrase "library automation," set some searching options, and search the full text of the IRLIST (information retrieval special-interest group) archives. After

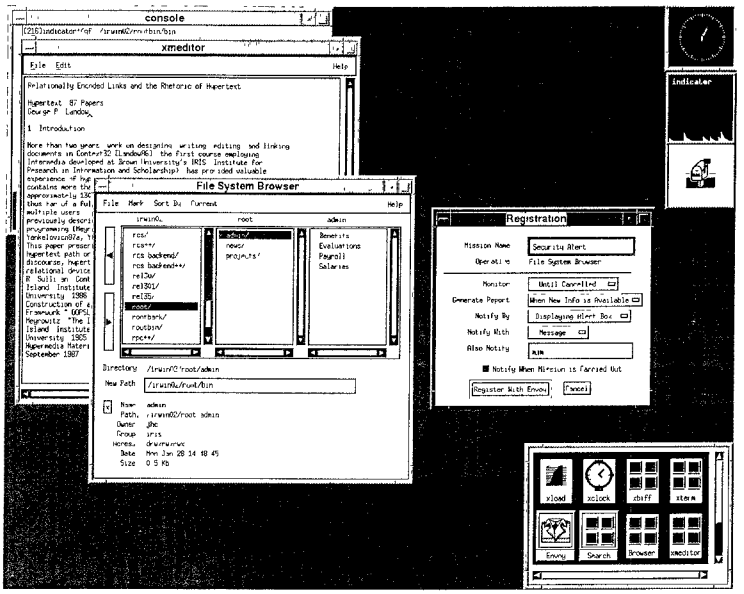


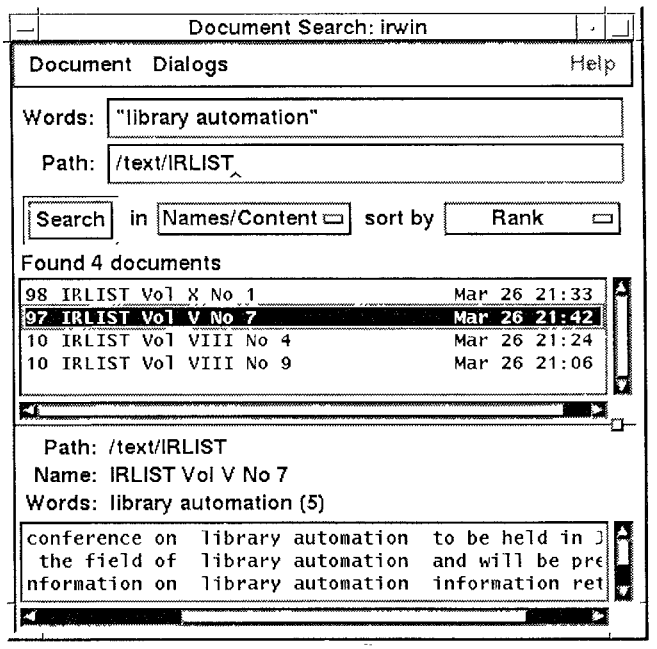
Fig. 4. An envoy-aware UNIX File System Browser operative with a directory "marked." The user has selected the "Register with Envoy" command and is now filling out a Registration Form.

examining all the occurrences of the phrase in the existing collection, the user decides to track the subject of library automation in the IRLIST and be alerted if any new information becomes available.

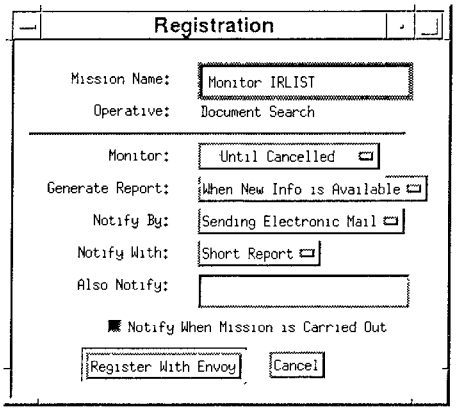
Since the mission is already specified—terms have been entered and options selected—the user picks the generic "Register with Envoy" command that is included in all envoy-aware applications. A Registration Form opens, presenting a number of choices (Figure 5). Following the scenario above, the default registration options would send the user a short report via electronic mail as oon as a new electronic issue of IRLIST was archived that contained the phrase "library automation." The mission to monitor the electronic publication would be named automatically (e.g., Mission: Find "library automation" in/text/IRLIST) and would persist indefinitely.

The user can dismiss the Registration Form dialog immediately, accepting the default settings, or the user can select a number of alternate registration options. The first registration option a user might change for a mission is the default mission name. A user might prefer a shorter or more personally meaningful name. If library automation is the only topic a user is interested in, then a mission name such as "Monitor IRLIST" might suffice.

For a combined operative such as Document Search, which can be used to run queries at scheduled times or can be used to monitor full-text databases, the user has the maximum number of alternate registration options available. In terms of scheduling, the user can specify dates for the mission to be carried out or the user can specify time intervals such as "every six hours," "three times a week," or "every two months." If the user opts for monitoring



(a)



(b)

Fig. 5. Registering a Document Search mission to monitor the IRLIST archive directory for the phrase "library automation."

(the default, when possible), the user can specify when to start and stop monitoring, as opposed to indefinite monitoring. The user can also specify that the mission should be carried out only until results are found, at which time the mission should be concluded.

In addition, the user can decide when and how he would like to be notified. Reports can be generated as soon as new information becomes available (the default), at any time interval the user chooses, or on any specific dates the

user selects from a calendar. The user can choose to be notified with either a message or a short report (described below). One of these can be sent to the user by way of informer applications, which may include electronic mail, alert boxes, or FAX messages. Finally, a user can opt to add other users to the notification list. These other users will be notified also with a message or short report when mission results are available.

Included in our design for the Registration Form, but not in the first prototype, is a notification option called "Notify by Finding Me." Users selecting this option would not have to decide ahead of time which informer the Envoy should use to notify them when the mission they are registering returns results. Since mission results are generated at unspecified times, the Envoy should help the information find the user by employing the informer most appropriate considering the time of day, the current status of the user, and the urgency of the mission.

When an operative has results to report, the Envoy notifies the user via an informer application such as electronic mail. The Envoy sends the user a short report, summarizing the mission results. For example, if the user has specified a mission using the Browser to monitor a shared directory, the Envoy would send a short report summarizing changes to the directory that occurred since the user's last short report.

As soon as a mission is registered with the Envoy, the mission is recorded in the Mission Summary table. This table, opened by clicking on the Envoy icon on the desktop, provides a comprehensive list of a user's active missions. Users can refer to the Mission Summary to find out the status of a mission, a mission's schedule, and the operative responsible for a mission. An entry is added to the Mission Summary table each time an operative reports results to the Envoy. A user who has been notified of new mission results, through electronic mail or some other informer, can find out additional details about the results by interacting with the Mission Summary. Although the short report that the Envoy sends as notification to the user may contain sufficient information, users will often want to see a more complete *interactive report*. Unlike short reports, interactive reports provide results in their "native" environment, giving users access to the full interactive capabilities of the operative user interface. When a user double-clicks on a mission result in the Mission Summary table, the Envoy launches the corresponding operative, passing the application the appropriate data to display. For example, if a user opens an interactive report concerning a mission to monitor a shared directory, the Browser interface would open, showing the contents of the shared directory with symbols indicating how items in the directory have changed (Figure 6).

In addition to opening interactive reports, users can also interact with the Mission Summary to cancel a mission, change the registration options for a mission, or review a short report associated with a mission. Users can also request an interim report by selecting the "Open Report" command from the Mission Summary window.

The Envoy user interface provides a single, consistent scheme for users to register missions. Coupled with this, the interface allows users to monitor

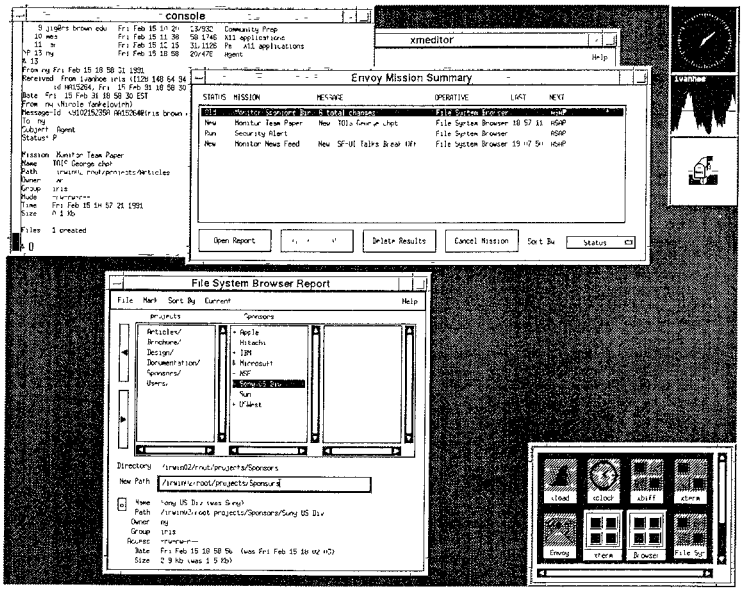


Fig. 6. By clicking on the Envoy icon (bottom right), the user has opened the Envoy Mission Summary table showing a list of his active missions (top center). The user selected the mission called "Monitor Sponsors Dir" from the table, clicked on the "Open Report" button, and is now examining the corresponding interactive report (bottom left). Notice that the Browser opens to show the contents of the monitored directory. All changes in the directory since the user last examined a report are indicated (+ for additions, − for deletions, and & for changes). In the bottom portion of the Browser window, the user can see which attributes of the selected document have changed.

and manage multiple missions easily. Automatic notification of mission results with informer applications allows information to actively seek out the user rather than forcing the user to seek out the information.

5. ENVOY ARCHITECTURE

Now we examine how our prototype Envoy Framework supports the User Interface section above by tracing the flow of data through the various system components.

Figure 7 represents data flow and channels of communication among the Envoy Framework's various components. In this example, the user is interacting with the full-text-retrieval operative, Document Search. As mentioned earlier, Document Search is a combined operative. Users can either schedule queries to be run at specific time intervals, or they can have Document Search monitor the file system, searching files as they are added or updated. In the example in Figure 7, the user schedules a Document Search query.

To interact with an Envoy, the user first specifies a mission (A). For example, the user might enter a full-text query by interacting with Document Search, pick "Register With Envoy," and fill out a Mission Registration Form. The operative frontend, the Document Search user interface in this example,

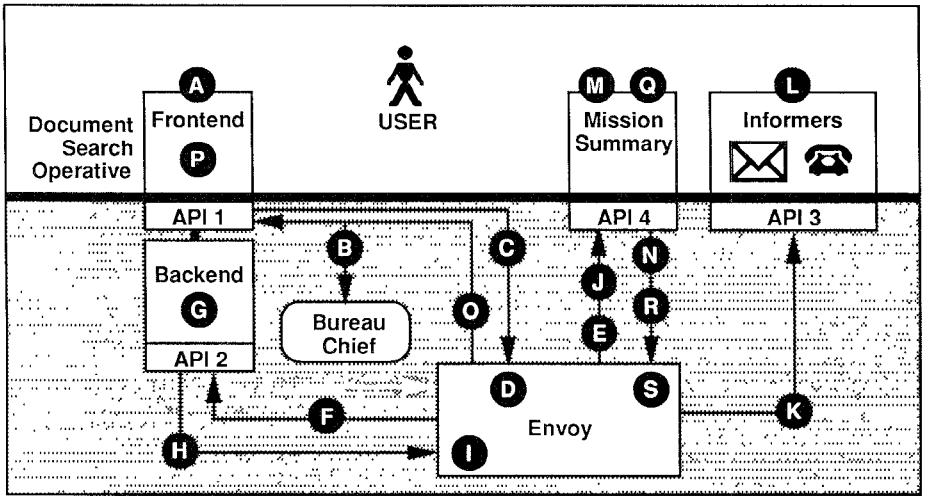


Fig. 7. Data flow and communication among the Envoy Framework components. The top portion of the diagram illustrates system components that the user interacts with directly.

encapsulates the mission data using a format that the backend can easily interpret. The operative contacts a Bureau Chief to find the location of the user's Envoy (B). If one does not exist, the Bureau Chief creates a new one. Now that the operative knows how to find the user's Envoy, it conveys both the encapsulated mission data and the mission registration information to the Envoy (C).

Assuming that in the mission registration the user requested the query to be run at scheduled intervals, the Envoy computes the next launch date and time based on the mission registration information, then adds a new mission entry to its list of registered missions (D). The mission entry consists of the encapsulated mission data and the mission registration information, which includes the next launch date and time. The Envoy sends a message to the Mission Summary with information about the newly registered mission, and the Mission Summary updates the user's list of missions accordingly (E).

At the scheduled time, the Envoy contacts the operative backend and passes it the encapsulated mission data (F). The operative backend parses the encapsulated mission data, interpreting it as if the user had just entered the data interactively. The mission is then carried out (G). In the case of the Document Search operative, the user's query would be executed.

After performing the mission, the operative returns data to the Envoy for constructing a message, a short report, and an interactive report (H). The Envoy adds the message and short report to the mission entry and saves the interactive report data in a file, maintaining a reference to it from the mission entry (I). From the mission entry, the Envoy determines what type of notification the user has requested.

The Envoy updates the Mission Summary, adding a new mission report entry (J). The message is always displayed in these entries. The Envoy

launches the appropriate Informer, electronic mail for example, and passes it either the message or the short report, depending on the user's specification (K). The Informer notifies the user of the results (L).

At this point, the user might decide to view an interactive report by double-clicking on the mission report entry in the Mission Summary (M). The Mission Summary informs the Envoy which report the user selected (N). The Envoy launches the operative frontend, for example a Document Search interface, and passes the interactive report data contained in the file (O). On receiving the interactive report data, the operative frontend interprets the data and displays the results (P). In the case of Document Search, the results of the registered query would be displayed, and all the documents retrieved since the last report would be highlighted.

At this stage, the user may no longer need the mission to be carried out, and therefore the user cancels the mission from the Mission Summary (Q). The Mission Summary informs the Envoy about the canceled mission and updates its list of user's missions (R). The mission entry in the Envoy's list of registered missions is also updated to reflect that the status of the mission is now canceled (S). The mission entry will be deleted only from this list if the user deletes all the results displayed in the Mission Summary associated with the mission.

If the user had specified in the mission Registration that Document Search should monitor the file system for the search terms rather than run the query at scheduled intervals, the flow of events and data would be altered slightly. Instead of contacting the Document Search backend at scheduled intervals, the Envoy would contact the Document Search backend only once, sending it the mission data. It is then the application's responsibility to do the monitoring. If the user cancels a mission, the Envoy would inform the Document Search backend, and the backend would delete the mission from its list of items to monitor.

The components of the Envoy Framework can be distributed across a LAN as shown in Figure 8. The figure shows a user "muru" viewing Mission Summary from one machine, the Envoy running on a different machine, and Document Search mission being carried out on a third machine. By making agent functionality available on a network-wide basis, users are not constrained to the resources available on their personal machine. Their Envoy can locate them regardless of where they are working on the network.

5.1 APIs (How to Become Envoy Aware)

The architecture we have defined for Envoys encapsulates the core agent functionality by defining application programmer interfaces (APIs), which interact with the main Envoy components and the operating system. Developers may upgrade their software to become envoy-aware applications by adhering to the APIs. This approach is similar to the way developers create (cut/copy/paste) compliant Macintosh applications and also similar to the way linking functionality is abstracted out in Intermedia, a hypermedia system developed at IRIS [34]. In IRIS, the core linking functionality (a database of document IDs, link endpoints, link property sheets, navigation

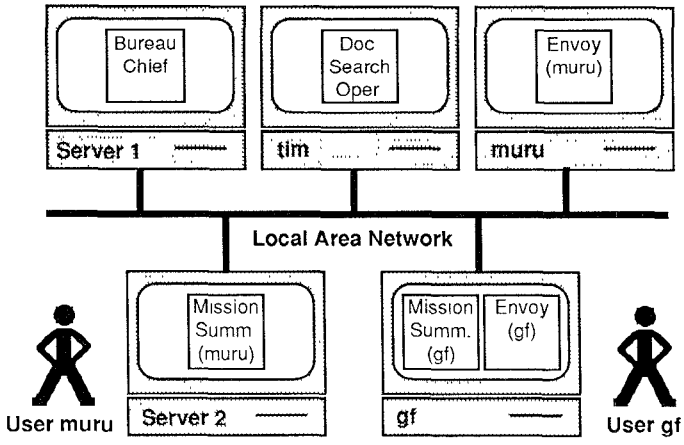


Fig. 8. Distributed Envoy components.

aids, etc.) is integrated into the operating system. End-user applications participate in this functionality by implementing a linking protocol, thereby becoming link aware. In a similar way, applications become envoy aware by adhering to an interprocess envoy/operative protocol, which is part of our decentralized, distributed Envoy architecture. Applications that implement the envoy/operative protocol inherit agent functionality, which includes tracking missions, launching missions at scheduled times, reporting results of missions to users by invoking informer applications, and continuously updating information in the Mission Summary to keep users well-informed of the status of their missions.

Following our design philosophy, we have defined two APIs that operative programmers must implement to make their applications envoy aware: one API that informer programmers must implement and one that developers who wish to create alternate mission summaries must implement (see Figure 7). These APIs are implemented as object-oriented classes. A table accompanies each API description below to summarize the set of calls available to the developer and the set of calls the developer is required to implement to create an envoy-aware application.

5.1.1 *Operatives.* It is a fundamental requirement in becoming envoy aware that operatives function in both an interactive and noninteractive mode. In the interactive mode, users converse with the application through either a command line interface or a graphical interface. Operatives must allow users to define one or more tasks interactively. In order to carry out tasks for the user behind the scenes, an operative must be able to run in a noninteractive mode, accepting raw data input from the Envoy and performing the mission within a background process.

Some operatives such as our File System Browser and Document Search applications are *continuous*. That is, the server process is constantly running in a noninteractive mode, accepting mission requests from interactive client

processes. Alternatively, operatives can be *discrete* and may be invoked by the Envoy either in an interactive or a noninteractive mode. When an Envoy calls on a discrete operative to carry out a mission, the Envoy launches the operative and the operative performs its task, terminating when the task is complete.

To make an operative frontend envoy aware, the operative writer has two primary tasks. First, the frontend must be able to define a mission to be conveyed to the Envoy and later to the operative backend. Once a mission has yielded results, tbe frontend must be able to display mission results in the form of an interactive report.

API 1. Envoy Interface to Operative Frontend

Available Calls	To Be Implemented
RegisterWithEnvoy( )	DoInteractiveReport( ) DoReceiveData( )

To send mission data, an operative writer makes a function call, RegisterWithEnvoy( ). The operative writer must encapsulate the data and parameters constituting the user's mission. There are no specific requirements for how to encapsulate the data. The operative writer can either send the data as a raw stream of bytes or as an encoded string. The Envoy will pass the data packet, untouched, to the operative backend at the appropriate time. RegisterWithEnvoy( ) first connects to the Bureau Chief running at a predetermined machine and port number. Contact with the Bureau Chief is made to obtain the current address of the user's Envoy and to validate that the specified operative has been installed into the framework. The Registration Form is presented to extract the user-specified registration data before locating and communicating this mission information to the user's Envoy.

When a user chooses to see an interactive report from the Mission Summary application, a message is sent to the Envoy that will then contact the operative frontend and send the necessary data to the application by calling DoReceiveData( ). By invoking this method, we allow an application to do any necessary processing before receiving a new report. For example, if an application is already displaying an interactive report, the application can save the report data before displaying data corresponding to a new report. The API receives the new data and calls the DoInteractiveReport( ) method, implemented by the operative developer. This method is responsible for parsing the interactive report data and presenting the information in a manner consistent with the application's user interface.

API 2. Envoy Interface to Operative Backend

Available Calls	To Be Implemented
InformEnvoy( ) InvalidMission( )	ReceiveNewMission( ) GetMissionList( )* CancelMission( )

* Only necessary to implement for continuous operatives.

When operatives function in a noninteractive mode, the Envoy calls the developer-implemented ReceiveNewMission( ) function when the time arrives for the operative to perform a specified task. The Envoy passes the operative backend a unique mission ID, which is formed by concatenating a user's identification number and a number assigned by the Envoy. The encapsulated mission data and a return address to locate the Envoy are also passed as parameters to the ReceiveNewMission( ) function.

When the operative completes the mission and has results to pass back to the user, the backend calls InformEnvoy( ), which contacts the user's Envoy and returns the results of the mission. It identifies the mission ID and a unique result ID, since one mission can generate multiple results (reports). Alternately, the backend can choose to notify the Envoy that new results are available, but not send the results until the user chooses "Open Report" from the Mission Summary.

The operative writer needs to generate three types of reports: an interactive report, a short report, and a message. The interactive report may be encapsulated in the same manner as the mission data. This data will get passed to the operative frontend when a user requests to view an interactive report. The short report or the message may be used by any of the informers to notify users of results. The message is always displayed in the Mission Summary. The InformEnvoy( ) call first tries to contact the user's Envoy at the last known address. If it cannot establish a connection, it contacts the Bureau Chief to verify the address or retrieve a new address.

At any time, an operative may receive a request from the Envoy to discontinue a mission. The Envoy will pass the unique mission ID to the operative. Continuous operatives will use the mission ID to clean up any internal data structures they have used during the lifetime of the mission. Once canceled, an acknowledgment message is sent back to the Envoy to report a successful cancellation of the mission.

Continuous operative writers must implement a few additional methods to ensure that the operative receives a mission request and that it is working on a valid mission. Because the server may be carrying out multiple missions at any given time, it is necessary for the Envoy Framework to validate the operative's known missions during consistency checks (e.g., whenever the continuous operative is restarted). If a system failure occurs and there is no mission validation, the operative may lose track of missions. Consequently, continuous operative writers must implement a method, GetMissionList( ), which generates a linked list of known missions by mission ID. This list is used in the validation process. At some point in the life span of a mission, the mission may become invalid. For example, a user might revoke read privileges for a directory being monitored by the File System Browser. Since the directory can no longer be monitored, the mission is rendered invalid. The operative notifies the Envoy that the mission is no longer valid by calling InvalidMission( ). This function will change the status of the mission to Invalid in the Mission Summary, and an error report specified as an argument in the function call will be presented to the user. Depending on the severity of the circumstances that caused a mission to become invalid, the

operative backend can specify that the mission should automatically be canceled by setting a parameter to InvalidMission( ).

5.1.2 *Informers.* Application developers creating Informers need to adhere to a very simple interface to make these applications envoy aware.

API 3. Envoy Interface to Informer

Available Calls	To Be Implemented
GetUserAttribute( )	InformUsers( )

Informer writers need to implement one function call, InformUsers( ), to become envoy aware. This function is supplied with three parameters: a primary user, a list of other users to be notified, and the mission report, which is always in ASCII.

If an informer requires additional information about a user, the Informer writer makes a call to the utility routine GetUserAttribute( ), which is provided to retrieve a specified user attribute. For example, a FAX informer would need a user's FAX telephone number. The routine takes as parameters the user login name and an attribute. It contacts the Bureau Chief to get the address of the user's Envoy. Next, it establishes a connection to the Envoy and retrieves the value of the specified attribute.

5.1.3 *Mission Summary.* The Envoy Framework provides a flexible and extensible interface for tracking and managing a user's set of registered missions. In the current prototype there is one application that provides this functionality: the Mission Summary.

API 4. Envoy Interface to Mission Summary

Available Calls	To Be Implemented
CancelMission( )	InstallMission( )
ReregisterMission( )	ChangeStatus( )
OpenReport( )	AddResults( )
DeleteResults( )	

Anyone developing an alternative mission summary application must adhere to three function calls: InstallMission( ), ChangeStatus( ), and AddResults( ). The InstallMission( ) call informs the application that the user has registered a new mission with the Envoy. The ChangeStatus( ) call indicates that the status of a mission has changed. For example, a mission that was scheduled to be launched at a specified time is currently running. The AddResults( ) call specifies that there is a new report for a mission. It is the responsibility of the application to decide how to present this information to the user.

API 4 also defines four function calls: CancelMission( ), ReregisterMission( ), OpenReport( ), and DeleteResults( ). The CancelMission( ) call sends the delete mission information (e.g., the unique mission ID) to the user's Envoy. The Envoy in turn, locates the operative and relays the cancel-mission request.

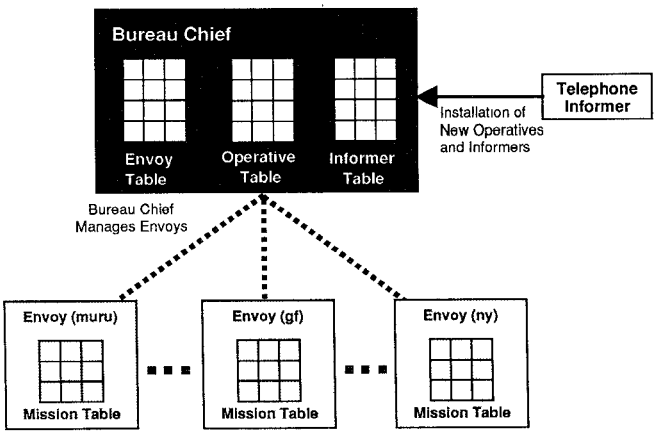


Fig. 9. Bureau Chief overview.

ReregisterMission( ) opens the mission registration dialog and sends the user-specified registration data to the user's Envoy; the new mission information is installed. The OpenReport( ) call conveys the user's desire to see an interactive report of a mission. To view an interactive report, OpenReport( ) requires the unique mission ID and result ID, which are provided when new results are received, via AddResults( ). The user's Envoy is contacted to retrieve the interactive report data and pass the data to the proper operative frontend, launching one if necessary, and presenting the report. When a user wishes to remove mission results, DeleteResults( ) will contact the Envoy which, in turn, removes the data associated with the specified mission ID and result ID.

The four APIs presented in this section represent an open architecture that allows many applications to participate in the Envoy framework.

5.2 Managing the Envoy Framework

Since the Envoy Framework architecture encompasses multiple applications, machines, and users, the model uses a Bureau Chief to coordinate the interactions of the various system components. The Bureau Chief is responsible for maintaining a database of all Envoys and envoy-aware applications as shown in Figure 9.

An Envoy Table is used for recording the port number and the machine on which each user's Envoy is running. Likewise, the Bureau Chief records information about envoy-aware applications in the Operative and Informer Tables. This information includes the path and name of the executable as well as what machines the executable will run on. The Bureau Chief process runs on a predetermined port at all times and provides three primary services: an Envoy lookup directory, envoy-aware application installation, and propagation of newly installed operatives to Envoys.

The Envoy lookup directory is consulted when a user registers a mission. At this time, the Bureau Chief is contacted to determine the location of the user's Envoy. If the Bureau Chief does not find an entry in the Envoy Table, it instantiates a new Envoy by selecting a machine with a relatively low load average, launching a new Envoy, and updating the Envoy table.

The Bureau Chief's second major service includes managing the application installation process. An application developer installs a new informer or operative into the Envoy framework by filling out a system-defined Installation form. In the form, the application developer specifies the application's name and location, whether it is an informer or an operative (distinguishing between continuous and discrete), and the machine architecture required to run the application. The machine architecture is important as it tells the Envoy framework the types of hardware that the operative or informer executable will run on. The developer can optionally indicate that the operative/informer should preferably be run on a specified machine or even indicate that the operative/informer must run on a specific machine. The installation information in the form is passed on to the Bureau Chief, which will perform the necessary consistency checks, validate the operative or informer, and assign it a unique identifier within the Envoy Framework. The application developer is notified immediately of any installation errors. Such errors include unknown machine types or machine names and inaccessible executables. The Bureau Chief then updates the Operative Table or Informer Table, adding the new information to the database.

The Bureau Chief propagates the information about the newly installed envoy-aware application to the Envoys. This propagation occurs on a regular basis and allows consistency among Envoys without interruption of Envoy services. The Bureau Chief frees users from being tied down to any one machine, building on our philosophy that information should find the user rather than vice versa.

5.3 Reliability and Resilience

Although (1) our system is a prototype and (2) we have not focused our efforts on creating a failsafe system, we have identified a number of important reliability issues. By reliability, we mean that once a user has delegated a mission to the Envoy, the system should make every reasonable effort to complete the mission. Failure of a Bureau Chief, Envoy, or operative are all possible obstacles to successful completion of a delegated mission.

When the Bureau Chief crashes, for example, routine operations such as operatives returning results and users viewing interactive reports can still be performed; however, the Envoy framework loses its lookup service capability and cannot register new missions. In addition, new operatives and informers cannot be installed into the framework.

Failure of an Envoy prevents the user from interacting with the Mission Summary, prevents operatives from returning mission results to the Envoy, and prevents scheduled missions from being launched. When an Envoy is restored, all of the user's missions must be reestablished, and overdue missions must be launched.

If an operative prematurely terminates, missions can no longer be performed, and results cannot be sent to the Envoys. Since continuous operatives maintain their own list of registered missions, a failure requires that these operatives reestablish all active missions when restarted and reprocess any missions that were incomplete when the failure occurred.

In our prototype, we focused most closely on the crucial task of ensuring mission persistence. Since a single mission may persist for an indefinite period of time, Envoys maintain mission information redundantly, both in memory and in a database. Whenever a new mission is registered or the status of a mission changes, the database information is updated. If an Envoy crashes, a user's missions can be reestablished by retrieving the information from the database. No user intervention is required to restart missions or to run missions scheduled to launch during the time the Envoy was unavailable.

5.4 Experience with the Prototype

The development of the Envoy Framework encompasses two distinct efforts: development of the Envoy core and upgrading of applications to adhere to the envoy/operative or envoy/informer protocols. All the Envoy core components except the Bureau Chief have been implemented. The implementation of the core as well as the operatives and informers are in C++ with X11R4 and Motif for Sun Sparcstations and Sun 4/110's running Sun OS 4.1. The prototype allows operatives and informers to carry out tasks on other UNIX-based systems such as MAC II's running A/UX. An earlier version used TCP/IP while the current version uses RPC++ (an object-oriented version of Sun's RPC) for communication among the various components. We present below some initial evaluations from operative developers, framework designers, and end users.

5.4.1 *Developer Evaluation.* To assist operative developers we published an "Operative Designers API Cookbook." The Cookbook provides detailed descriptions of the APIs for discrete and continuous operatives. Continuous operative developers adhere to the API by subclassing two classes, cEvCltOperative and cEvSvrOperative. For example, Document Search defines two subclasses: cDSCltOperative and cDSSvrOperative. After the initialization of the cDSCltOperative within the client's main( ) procedure, the developer calls the Operate( ) method. Within this method, the superclass determines if the frontend is to receive an interactive report. If so, the superclass invokes the DoInteractiveReport( ) method; otherwise it invokes the DoFrontend( ) method. After the initialization of the cDSSvrOperative, the superclass deals with servicing envoy-specific commands. For instance, when the superclass receives an RPC request to register a new mission, it invokes the application's ReceiveNewMission( ) method.

Although the prototype is too new for us to have gathered much feedback, our experience with upgrading applications into operatives thus far suggests that it takes a matter of days to implement the envoy/operative protocol.

As we had anticipated, the two most difficult aspects of accomplishing the upgrade involved deciding what application capabilities should constitute a

mission and deciding how to modify the frontend to support viewing of interactive reports. For example, in our File System Browser application, the developer had to decide whether a user who marked a directory would want to monitor only the contents of the directory or the contents of its subdirectories as well. The developer decided to support both possibilities and added a toggle button to the interface to allow users to choose between the two options.

In designing the Browser's interactive reports, the developer initially experimented with displaying a dialog box with a table summarizing all the changes to a monitored directory or file. The developer eventually decided to rely more heavily on the existing user interface design and prefixed file and directory names with plus, minus, and other symbols to indicate which items were created, deleted, moved, or changed since the user's last mission report. To indicate specifically what attributes of a file or directory had been modified, the developer appended the text "(was . . .)" to each attribute that had been changed (see Figure 6).

Based on this initial experience, we are optimistic that developers can adhere to the protocols we have designed with minimal time and effort that does not involve substantial redesign of their applications.

5.4.2 *Framework Evaluation.* For the few applications that have become envoy aware, we are beginning to assess the success of the framework in terms of the functionality that it provides compared with the functionality required by the envoy-aware applications. Abstracting functionality such as notification, tracking, and scheduling into the framework alleviates the need for application developers to providing this common functionality. While the prototype demonstrates that these services do reduce the work involved in building operatives, we are finding that additional functions should be provided to support mission management. Assisting developers in creating continuous operatives is one area in need of better management services. Currently, once a continuous operative is launched, the operative is required to handle more than one mission at a time. It is important that each mission is given adequate attention, and that not too much time is spent working on one mission while neglecting another or not responding to requests by the Envoy (e.g., registering new missions). For now, developers need to build this type of balanced work behavior into their applications manually. Since this is an issue all continuous-operative developers face, it would be more convenient if the Envoy framework provided a generalized mission management service.

The framework could also be expanded to provide continuous operatives with optimization services for handling concurrent requests and with consolidation services. If more than one user makes the same request, the operative should be able to consolidate these requests. Duplication of effort can be costly and can tie up valuable shared resources. Since each operative developer will have to handle consolidation, it would benefit the developer if the framework provided some help in this area.

In the early design stages, we decided to define an API between the Envoy and Mission Summary application. At first this seemed awkward and unnec-

essary; however, as the design unfolded, we realized that this API was needed to prevent the Mission Summary application from being the only means of accessing and modifying registered mission information. With the API, alternate mission summary applications can be created and plugged into the framework with very little effort. For example, a telephone-based Mission Summary has been proposed. Using this application, users would be able to call their Envoy and issue commands by selecting a sequence of numbers; a voice synthesizer could provide status information and read mission reports. Having this API allows for the telephone-based as well as other new mission summary applications to be developed.

Finally, one of the major tests of the framework has been in its use as the basis for an office automation application that implements a workflow model centered around electronic forms [13]. A few new operatives were designed to handle filling-in and routing of electronic forms. Also, the notion of Envoys was extended; and a new class of Envoys was designed: Workflow Envoys. These Workflow Envoys track and manage a series of related submissions in order to carry out a large workflow task (e.g., a travel report approval and reimbursement process). The framework proved to be a successful and quick means of prototyping this workflow model. The Workflow Envoys were modeled after the user Envoys and were able to be integrated into the framework by adhering to the existing APIs. The Envoy framework was modified so that users did not always have to explicitly register missions. Instead, missions might be registered automatically, when for example, an employee receives a work request from a manager. The Mission Summary Application was hooked up with the Workflow Envoys to allow users to access and track the progress of each workflow instance; no changes to the Mission Summary were necessary.

5.4.3 *User Evaluation.* The Envoy prototype has not yet been fieldtested, so we cannot fully judge the effectiveness of the system in helping users monitor shared workspaces and full-text databases. We have, however, collected suggestions for improving the user interface.

We found that users wanted access to more information within the Mission Summary application. Users were confused that missions were not separated from their associated results. Our initial concept was to make the Mission Summary as compact as possible so that users would not mind leaving it open on the screen. Based on initial feedback, however, our belief is that it would be better to use one line to identify the mission and then list each report individually as it arrives. Users would then be able to expand or collapse the list of reports to see different views of the data.

We also discovered that while using an operative, some users wanted to see all currently pending missions involving that operative. A Mini-Mission Summary available via a menu command inside the operative window might be one way to address this concern. This window might provide a context-dependent subset of the information found in the full Mission Summary.

Users also often wanted a detailed description of one of their registered missions. The current interface does not provide this ability; user's can

examine the mission's registration form but not the actual definition. That is, they can see the name of a mission, what type of operative is involved, and how frequently a mission is to be run, but they cannot get information about the specifics of the mission. For example, the user might want to find out that a selected mission involves searching for "library automation" in the IRLIST archive. Supplying a textual description of the mission in a dialog box would definitely be quite helpful. On the down side, this would place an additional burden on the developer whose job it would be to create English language descriptions of each mission.

6. CONCLUSIONS

We describe a model in which Envoys allow users to specify missions within applications that operate on heterogeneous machines across a local-area network. Envoys use operatives to carry out missions and use informers to notify users of the results of missions. In Section 3 we identify a set of characteristics and capabilities that we believe an agent system should exhibit. Below we evaluate our model on its conformance to these characteristics and capabilities.

The model supports event-triggered, time-triggered, and repetitive execution of tasks. Time triggering, which we also refer to as delayed execution, is internal to the Envoy process; however, most event triggering is handled at the operative level. For example, to monitor the file system for documents matching a user's query, Document Search triggers execution of the query when new files are added to the file system or existing files are modified. This way, the Envoy Framework does not have to determine all possible events that users might want to use as triggers for launching missions and does not have to be concerned with variations among differing machines and system configurations. In addition to event or time triggering, the Envoy framework supports repetitive execution of missions. For example, users can specify that they want a mission to be carried out three times a day for the next two weeks.

Our philosophy has been to introduce Envoy functionality seamlessly into the user's computing environment and to integrate Envoys with applications that users already employ to carry out their work. The agent functionality is embodied as a desktop service. Consequently, users only need to learn a few simple additional concepts to take advantage of Envoys. Consistency is emphasized throughout the system. For example, we provide a uniform user interface for registering missions, independent of any application, as well as a standard method of opening and viewing mission results.

The Envoy Framework's Mission Summary application provides a mechanism for the user to track and manage a set of registered missions. Our prototype demonstrates an extensible, decentralized architecture by abstracting agent functionality into a set of components and APIs. All of the message communications among the Envoy components have been abstracted to obviate the need for operative and informer developers to be concerned with this level of detail. For example, by separating out the mission notification, an

operative writer knows that users will be able to use any Informer application for reporting the results of missions. Operative developers are not burdened with implementing application-specific methods of communicating results to the user.

The Envoy architecture has been designed to support a distributed environment. A user can access an Envoy from any UNIX-based computer on the local-area network, and missions can be carried out on a computer other than the one the user is operating. The system is resilient enough to retain a user's set of registered missions through system and process failures.

Functionally, we would like to enhance the Envoy Framework to support pipelining of tasks and to work across a wide-area network. One approach might be to base the Envoy Framework on OMG's object management architecture [26], using their Object Request Broker technology. This architecture will support interoperability between applications on different machines in heterogeneous distributed environments. We also plan to assess the need and utility of both interagent communication and adaptability.

If a protocol similar to our envoy/operative protocol is integrated into standard operating environments, then users will be able to benefit from a host of agent-aware applications. Users will no longer have to be information seekers. Information will find the users.

REFERENCES

1. ADAMS, S. S. AND NABI, A. K. NeuralAgents: A frame of mind. In *OOPSLA '89 Proceedings* (Oct. 1989), ACM, New York, 139–149.
2. ANDERSON, R. H., AND GILLOGLY, J. J. Rand Intelligent Terminal Agent (RITA): Design philosophy. ARPA Order, 189-1, Rand, Santa Monica, Calif., Feb. 1976.
3. APPLE COMPUTER, INC. Project 2000–A Knowledge Navigator. (Videotape). Available from Apple Video Fulfillment Program, 1-800-627-0230, Mar. 8, 1988.
4. CHEN, F. F., PRAKASH, A., AND RAMAMOORTHY, C. V. The network event manager. In *Proceedings of the Computer Networking Symposium* (Washington, D.C., 1986).
5. COOMBS, J. H. Hypertext, full text, and automatic linking. In *SIGIR '90 Proceedings: 13th International Conference on Research and Development in Information Retrieval* (Brussels, Sept. 1990) ACM, New York, 1990, 83–98.
6. DIALOG INFORMATION SERVICES, INC. AP news reloaded as a single file; New fields and Dialog alert service added. *Chronology 18*, 4 (Apr. 1990).
7. DON, A., OREN, T., AND LAUREL, B. Guides 3.0. In *CHI'91 Video Proceedings* (New Orleans, La., Apr. 1991).
8. DOW JONES AND COMPANY, INC. *An Overview of DowVision*. DowVision, Princeton, N.J., 1990.
9. DROMS, R. E. Access to heterogeneous directory services. In *Proceedings of the IEEE InfoCOM '90 Conference* (Jun. 1990), IEEE, New York, 1990.
10. ELLIS, C. A., GIBBS, S. J., AND REIN, G. L. Groupware: Some issues and experiences. *Commun. ACM 34*, (Jan. 1991), 38–58.
11. ELLMAN, T. Explanation-based learning: A survey of programs and perspectives. *ACM Comput. Surv. 21*, 2 (Jun. 1989), 164–221.
12. FENTON, J., AND BECK, K. Playground: An object-oriented simulation system with agent rules for children of all ages. In *OOPSLA '89 Proceedings* (Oct.) ACM, New York, 1989, 123–137.
13. FITZMAURICE, G. Form-centered workflow automation using an agent framework. Masters thesis, Brown Univ., Aug. 1991.
14. FREEDMAN, B. Verity upgrades topic text manager. *PC Week* (Jun. 12, 1989).

15. GIBBS, S. J. LIZA: An extensible groupware toolkit. In *CHI'89 Conference Proceedings* (Austin, Tex., Apr. 1989) ACM, New York, 1989, 29–35.
16. GIFFORD, D. K., AND FRANCOMANO, A. M. An information system based upon programmable agents. In *CIPS Edmonton '90 Information Technology Conference*, (Edmonton, Alberta, Oct. 1990).
17. HAMMAINEN H., ELORANTA E., AND ALASUVONTO, J. Distributed form management. *ACM Trans. Inf. Syst. 8*, 1 (Jan. 1990), 50–76.
18. HEWLETT PACKARD. *HP NewWave Agent Guide*. Santa Clara, Calif, Oct. 1989.
19. KAHN, R. E., AND CERF, V. G. *The Digital Library Project· The World of Knowbots Vol. 1.* Corporation for National Research Initiatives, Mar 1988.
20. KAYE, A. R., AND KARAM, G. M. Cooperating knowledge-based assistants for the office *ACM Trans. Office Inf. Syst. 5*, 4 (Oct. 1987), 297–326.
21. LAI, K. Y., AND MALONE, T. W. Object lens: A "spreadsheet" for cooperative work *ACM Trans. Office Inf. Syst 6*, 4 (Oct. 1988), 332–353
22. LAUREL, B. Interface agents: metaphors with character. In *The Art of Human-Computer Interface Design*, B. Laurel, Ed. Addison-Wesley, Reading, Mass., 1990, 355–366.
23. MALONE, T. W., GRANT, K. R., LAI, K. Y., RAO, R., AND ROSENBLITT, D. Semi-structured messages are surprisingly useful for computer-supported coordination. *ACM Trans. Office Inf. Syst. 5*, 2 (Apr. 1987), 115–131.
24. MACDONALD B. A. Instructable systems. *Knowledge Acquisition 3*, 4 (Dec 1991), 381–420.
25. MINSKY, M. *The Society of Mind*. Simon and Schuster, New York, 1986.
26. OBJECT MANAGEMENT GROUP. *Object Management Architecture Guide*. Framingham, Mass., 1991.
27. OREN, T., SALOMON, G., KREITMAN, K., AND DON, A. Guides: Characterizing the interface. In *The Art of Human-Computer Interface Design*, B. Laurel, Ed. Addison-Wesley, Reading, Mass., 367–381.
28. PALANIAPPAN, M., AND FITZMAURICE, G. InternetExpress: An inter-desktop multimedia data-transfer service. *IEEE Computer 24*, 10 (Oct. 1991), 58–67.
29. RODDEN, T., SAWYER, P., AND SOMMERVILLE, I. Interacting with an active, integrated environment. *SIGSOFT 13*, 5 (Nov. 1988), 76–84.
30. SUN MICROSYSTEMS, INC. *DeskSet Environment Reference Guide*. Part Number: 800-4929-10, Mountain View, Calif., June 1990.
31. SUN MICROSYSTEMS, INC. *SunNet Manager Installation and User's Guide*. Part Number: 800-3481-10, Mountain View, Calif., Mar. 1990.
32. *Unix System Manager's Manual.* University of California, Berkeley, March 1984.
33. WEYER, S. A., AND BORNING, A. H. A prototype electronic encyclopedia. *ACM Trans. Office Inf. Syst. 3*, 1 (Jan. 1985), 63–88.
34. YANKELOVICH, N., HAAN, B. J., MEYROWITZ, N., AND DRUCKER, S. M. Intermedia: The concept and the construction of a seamless information environment. *IEEE Computer 21*, 1 (Jan. 1988), 81–96.

Received February 1991; revised June 1991; accepted March 1992